**Term Project Report**

**CS303L Algorithms and Data Structures**

**Sam Lazrak**

**Spring Semester 2018**

**Objectives:**

In this term project we will explore dynamic programming. The point of this exercise is to have you the student take on a programming assignment that is less scripted than the ones done during the lab work. For this project, we will implement Matrix Chain Multiplication as discussed in the text, Sections 15.2-15.3.

You may choose to work on this term project in teams of two or three students, or to work on it individually. If you desire to work as a team, send an e-mail to the instructor stating the students who will be working together. In a team approach, all of the team members’ names should be placed on the written report, and each member of the team must submit a copy of the report into Canvas for grading.

First, create a MATRIX-CHAIN-MULTIPLICATION class and include the bottom-up approach class methods MATRIX-CHAIN-ORDER and PRINT-OPTIMAL-PARENS. The pseudocode for these two algorithms are given below. Also, you will need a recursive method MATRIX-CHAIN-MULTIPLY, as outlined in Exercise 15.2-2.

Next, add to the class the top-down approaches RECURSIVE-MATRIX-CHAIN and MEMOIZED-MATRIX- CHAIN together with its helper method LOOKUP-CHAIN. (You may find that you want a method RECURSIVE-CHAIN-MULTIPLY as well. Which codes to write is largely up to you.)

Write a driver program that creates six matrices, A1 through A6 of the dimensions given in the example shown in Figure 15.5. Fill the matrices with random integers on the interval (0, 100).

Use a bottom-up approach to compute the product matrix A which is the result of multiplying all of the six input matrices together. Instrument your code so that you can measure the amount of time taken to compute the product matrix A.

Then use a top-down approach to compute the product matrix A using a recursive approach. Again, instrument your code to record the amount of time taken on the top-down recursive approach.

Finally, use the top-down memoized approach to compute the product matrix A and record the time required. Note the absence of the auxiliary array s in the top-down approach.

When you have written and run all of the code, and observed the running times, write up your observations about the amount of running time and space required. Contemplate the advantages to learning about studying an algorithmic approach and learning how to make programming as efficient as possible.

Submit your report in the format of a Lab Report into Canvas by the due date given.

Exercise 15.2-2: Matrix Chain Multiplication

Give a recursive algorithm MATRIX\_CHAIN\_MULTIPLY (A,s,i,j) that actually performs the optimal matrix-chain multiplication, given the sequence of matrices <A1, A2, ..., An>, the s table computed by MATRIX\_CHAIN\_ORDER, and the indices i and j. (The initial call would be MATRIX\_CHAIN\_MULTIPLY (A,s,1,n)).

MATRIX\_CHAIN\_MULTIPLY (A,s,1,n)

if n = 1 Solution hints:

return A1

• This is a recursive procedure, which passes in the parameters I and j which tell which sub-

else if n-1 = 1

chain of matrices in the chain A1 to An are to be multiplied.

return MATRIX\_CHAIN\_MULTIPLY (A1, An)

• It has two base cases: if the number of matrices in the sub-chain is 1, or if the number of

else matrices in the sub-chain is 2.

return MATRIX\_CHAIN\_MULTIPLY (

If only one matrix is to be “multiplied,” just return that matrix.

MATRIX\_CHAIN\_MULTIPLY (A,s,1,s[1,n]),

If two matrices are to be multiplied, call a helper routine to perform matrix multiplication

MATRIX\_CHAIN\_MULTIPLY (A,s,s[1,n] + 1, n) ) (you might have to write this).

In all other cases, recur to the first part of the sub-chain multiplied by the second part of the sub-chain. Here you will use the values in the s matrix to help you determine which “inner parts” of the sub-chain should be multiplied next.

**Source Code:**

**package TermProject;**

**public class Matrix {**

**private int rows;**

**private int columns;**

**private int[][] matrix;**

**public Matrix(int r, int c){**

**this.rows = r;**

**this.columns = c;**

**this.matrix = new int[r][c];**

**}**

**public int getRows(){**

**return rows;**

**}**

**public int getColumns(){**

**return columns;**

**}**

**public void setValue(int row, int col, int value){**

**matrix[row][col] = value;**

**}**

**public int getValue(int row, int col){**

**return matrix[row][col];**

**}**

**}**

**package TermProject;**

**import java.io.IOException;**

**import java.util.Scanner;**

**public class MatrixChainMultiplication {**

**private static int[] p = { 30, 35, 15, 5, 10, 20, 25 };**

**private static int[][] m;**

**private static int[][] s;**

**private static Matrix c;**

**private static Matrix[] array;**

**public static void matrixChainOrder() {**

**int n = p.length - 1;**

**m = new int[n + 1][n + 1];**

**s = new int[n + 1][n + 1];**

**for (int i = 1; i <= n; i++)**

**m[i][i] = 0;**

**for (int l = 2; l <= n; l++) {**

**for (int i = 1; i <= n - l + 1; i++) {**

**int j = i + l - 1;**

**m[i][j] = Integer.MAX\_VALUE;**

**for (int k = i; k < j; k++) {**

**int q = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];**

**if (q < m[i][j]) {**

**m[i][j] = q;**

**s[i][j] = k;**

**}**

**}**

**}**

**}**

**}**

**private static String printOptimalParens(int i, int j) {**

**if (i == j)**

**return " A" + i + " ";**

**else**

**return "(" + printOptimalParens(i, s[i][j]) + printOptimalParens(s[i][j] + 1, j) + ")";**

**}**

**public static void printOptimal() {**

**System.out.println(printOptimalParens(1, p.length - 1));**

**}**

**public static Matrix recursiveChainMultiply(Matrix[] A, int t, int i, int j) {**

**if (i == j) {**

**return A[i];**

**} else {**

**Matrix leftChain = recursiveChainMultiply(A, t, i, s[i][j]);**

**Matrix rightChain = recursiveChainMultiply(A, t, s[i][j] + 1, j);**

**matrixMultiply(leftChain, rightChain);**

**return c;**

**}**

**}**

**private static void matrixMultiply(Matrix A, Matrix B) {**

**if (A.getColumns() != B.getRows()) {**

**System.out.println("CAN NOT MULTIPLY");**

**} else {**

**c = new Matrix(A.getRows(), B.getColumns());**

**for (int i = 0; i < A.getRows(); i++) {**

**for (int j = 0; j < B.getColumns(); j++) {**

**c.setValue(i, j, 0);**

**for (int k = 0; k < A.getColumns(); k++) {**

**c.setValue(i, j, (c.getValue(i, j) + A.getValue(i, k) \* B.getValue(k, j)));**

**}**

**}**

**}**

**}**

**}**

**public static void memoizedMatrixChain() {**

**int n = p.length - 1;**

**m = new int[n + 1][n + 1];**

**s = new int[n + 1][n + 1];**

**for (int i = 1; i <= n; i++) {**

**for (int j = i; j <= n; j++) {**

**m[i][j] = Integer.MAX\_VALUE;**

**}**

**}**

**lookupChain(1, n);**

**}**

**public static int lookupChain(int i, int j) {**

**if (m[i][j] < Integer.MAX\_VALUE) {**

**return m[i][j];**

**}**

**if (i == j) {**

**m[i][j] = 0;**

**} else {**

**for (int k = i; k <= j - 1; k++) {**

**int q = lookupChain(i, k) + lookupChain(k + 1, j) + p[i - 1] \* p[k] \* p[j];**

**if (q < m[i][j]) {**

**m[i][j] = q;**

**s[i][j] = k;**

**}**

**}**

**}**

**return m[i][j];**

**}**

**public static void menu(Scanner input){**

**try {**

**start(input);**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**private static void start(Scanner input) throws IOException {**

**System.out.println("Options 'default' or an integer. ");**

**System.out.print("How many Matrices?: ");**

**try{**

**getDimensions(input);**

**}**

**catch(NumberFormatException e){**

**System.out.println("\nINVALID INPUT\n");**

**start(input);**

**System.exit(0);**

**}**

**finally{**

**input.close();**

**System.out.println();**

**}**

**long timeLTR = System.nanoTime();**

**timeLTR = System.nanoTime() - timeLTR;**

**long timeMCO = System.nanoTime();**

**matrixChainOrder();**

**timeMCO = System.nanoTime() - timeMCO;**

**long timeMMO = System.nanoTime();**

**memoizedMatrixChain();**

**timeMMO = System.nanoTime() - timeMMO;**

**long timeOpt = System.nanoTime();**

**timeOpt = System.nanoTime() - timeOpt;**

**printOptimal();**

**System.out.println("Matrix chain order runtime took " + timeMCO + " nanoseconds.");**

**System.out.println("Memoized matrix chain runtime took " + timeMMO + " nanoseconds.");**

**System.out.println("Multiplying in order of left to right took " + timeLTR + " nanoseconds.");**

**System.out.println("Multiplying in optimal order took " + timeOpt + " nanoseconds.");**

**System.out.println("Memoized matrix and multiplying total: [" + (timeMMO + timeOpt) + "] nanoseconds.");**

**}**

**private static void getDimensions(Scanner input) throws IOException {**

**String in = input.next();**

**if(in.contains("default")){**

**System.out.println("[30x35, 35x15, 15x5, 5x10, 10x20, 20x25]");**

**}else{**

**int n = Integer.valueOf(in) + 1;**

**p = new int[n];**

**System.out.print("Enter the first dimension of Matrix A(1): " );**

**int dimension = input.nextInt();**

**p[0] = dimension;**

**for(int i = 1; i < n; i++){**

**System.out.print("Enter the second dimension of Matrix A(" + i + "): ");**

**dimension = input.nextInt();**

**p[i] = dimension;**

**}**

**}**

**System.out.println("Computing... ");**

**}**

**@SuppressWarnings("unused")**

**private static void createMatrices() {**

**int n = p.length-1;**

**array = new Matrix[n];**

**for(int i = 0; i < n; i++){**

**array[i] = new Matrix(p[i], p[i+1]);**

**for(int row = 0; row < p[i]; row++){**

**for(int col = 0; col < p[i+1]; col++){**

**int value = (int) (Math.random() \* 5) + 1;**

**array[i].setValue(row, col, value);**

**}**

**}**

**}**

**}**

**public static String printMatrix(Matrix in){**

**String s = new String();**

**s = in.getRows() + " x " + in.getColumns() + "\n" + "\n";**

**for(int i=0;i<in.getRows();i++)**

**{**

**s = s+" ";**

**for(int j = 0; j<in.getColumns();j++)**

**{**

**s = s+in.getValue(i, j)+" ";**

**}**

**s = s+"\n";**

**}**

**return s;**

**}**

**public static void main(String[] args) {**

**createMatrices();**

**for(int i = 0; i < p.length-1; i++){**

**System.out.print(printMatrix(array[i]));**

**}**

**}**

**}**

**Output:**

**Dynamic bottom-up: 4.808539 ms**

**Memoized top-down: 1.179745 ms**

**Standard top-down: 1.399858 ms**

**Reference approach: 7.642849 ms**